DECSYSTEM 20

ERROR DETECTION, RECOVERY AND REPORTING REFERENCE MANUAL

Order No. EK-SEDRR-RF-001

digital

ERROR DETECTION, RECOVERY AND REPORTING REFERENCE MANUAL

Order No. EK-SEDRR-RF-001

digital equipment corporation · maynard. massachusetts

First Printing, February 1976

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

CONTENTS

PREFACE

"To err is human. To really foul things up takes a computer."

Ever since the birth of computers, these collections of solder, wire, silicon, and germanium have borne the brunt of many jokes. Most of these jokes are about what happens when things go wrong. Even the most knowledgeable people "inside the industry" quietly laugh when computers make mistakes. These are the people who know the inner workings of computers, and who often, unknowingly, refer to computers as "he" or "she". Much information is available concerning how computers do things right, but little is told about what really happens when computers do things wrong. What happens when a DECsystem-20 makes a mistake is the subject of this manual.

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

At first, computers were designed and operated to solve problems for one person at a time. This person was knowledgeable in the workings of the computer and its programs. Usually, when things went wrong, the only visible indication was either smoke or a very wrong answer. Since these early computers were most frequently used for long term research projects, the identification and correction of problems was not a pressing issue. If a problem took 3 days or longer to correct, only one person at most was unhappy, and the new invention or theory would be discovered a little later then otherwise expected.

As the usefulness and complexity of computer systems increased so did the dependency on the computer's output. Computers were being used to design new products, maintain inventory control, keep accounting records for whole companies, and produce employee's paychecks. Here, errors meant lost production time, shortages of critical parts, serious errors in a company's financial reports, and late paychecks. When errors were detected, the result was often lengthy "re-runs" of jobs after the problem was identified and corrected by the computer system maintainers. This identification process involved special test equipment and programs, and while the system was "down" no useful work was accomplished.

The computer systems of today are extremely complex, highly-sophisticated packages capable of serving hundreds of people at the same time, doing many different tasks. But the computers still make mistakes or break down. However, today the computer system recognizes its own errors or breakdowns. The system (both hardware and software) has been specifically designed to try to recover from its problems and provide system maintainers with detailed information regarding the errors or malfunctions. If the system is unable to recover from the error, the next goal of the system is to minimize the effect of the error. By doing this, only the fewest number of jobs or users are stopped from completing their tasks, and the rest may run to successful completion.

The development of more effective and defensive error detection, recovery, and reporting has played a major part in the development of today's system because people have become more and more dependent on computers to provide fast, accurate answers to their questions.

1.2 ERROR CATEGORIES

In any discussion about error recovery , a brief description of the errors likely to be seen is first necessary. These errors to be detected by the system can generally be divided into three basic types:

1. User programming errors

2. Operating system or monitor errors

3. Machine failures

1.3 USER PROGRAMMING ERRORS

Although the identification and repair of user programming errors is not the responsibility of field service engineers, you should be aware that these errors occur and understand how they affect the entire system. For this reason, only brief descriptions of these errors and their general handling procedure are included here.

1.3.1 Violation of System Architecture

There are several types of user programming errors. One is the violation of the architecture or the basic design of the system. The programmer must follow the rules set by the designers of the system if he is to use the system to get his job done. The best example of this basic system design or architecture is the machine's instruction set. Certain instructions may not be allowed to be executed by a user program but are reserved for the programs which both serve and control the users. These programs are called the operating system or monitor and the reserved instruction may do I/O functions or control the allocation of core memory. If a user program attempts to execute one of these reserved instructions or attempts to execute an instruction the machine doesn't understand, the user has violated the system architecture and the error will be caught.

1.3.2 Data Programming Errors

Another type of user-programming error involves data programming which may include either using the wrong format for data or incorrectly handling the input/output resources. An example of wrong data format might be incorrect specification of record length or operations on data using the wrong assumptions about the data's layout. Incorrect handling of the input/output devices might result from trying to do data input from a device, such as a paper-tape punch or line printer, that can only perform output.

1.3.3 Storage Allocation Errors

Still another type of user-programming error is an attempt to use or address more storage space than has been allocated or reserved for the program. This might even include trying to access more storage than is physically present on the system. This would usually occur only in programs which attempt to increase their storage capacity after the program has started its run. This class of error may also include attempting to access I/O devices which the program has not reserved or which may not exist on the system.

1.3.4 Control and Recovery of User Programming Errors

Control and recovery for user programming errors is accomplished by both the system hardware and operating system software. In the case of violation of system architecture, the CPU will most likely notice the problem and alert the system software via an interrupt or perhaps a trap. In any case, the user program is prevented from violating the rules and the offending job is automatically stopped and given an error message describing the violation. Data programming errors are usually caught by the data management section of the operating system as it performs the input/output request. Here the monitor may detect that the wrong type device has been selected, the requested data is not present, or that the data is different than that which the user expects. In cases where data for several different users resides on the same or similar devices (such as disk packs) the monitor will also check that the user is accessing his own data or is not trying to modify someone else's data. Usually errors of this type will also cause early termination of the user's job. The user's program may, however, include special routines to attempt to figure out what went wrong and correct the problem. In the event the operating system or the hardware or both detect a user trying to access storage space or devices not allocated to him, control is immediately transferred to the monitor and the user's job is stopped. The monitor may, in some cases, attempt to allocate the additional storage the user tried to access. If the monitor is successful, the user's program will be continued; otherwise the appropriate error message will be given and the user's job cannot be continued (will be aborted).

Almost all user programming errors will result in that user's job being stopped. The rest of the users continue to run and in this manner the system has minimized the effect of these errors.

1.4 MONITOR ERRORS

The second cause of errors to be considered is the operating system or monitor. Here full identification of the error is sometimes much more complex than identification of user programming errors. Again the correction of pure monitor errors is not the responsibility of field service engineers, but they must be aware of much more information about this class of errors because many of the errors may be caused by intermittent hardware problems. Field service engineers should be capable of discussing monitor errors with software specialists to determine if software or hardware is the cause of the failure.

Monitor or system software errors can be attributed to three major causes:

1. Bad programming initially

2. Unexpected error combinations

3. Undetected errors or outside errors

1.4.1 Bad Programming

Operating systems or monitors are developed by highly competent systems programmers who have an intimate knowledge of both the system hardware and software architecture; but they are still human and sometimes make mistakes. Most of these mistakes occur when the system designer doesn't consider each possible eventuality when developing the system. In some cases the programmer may consider how a system may arrive at a given point but he may not know how to get out of the point. This may occur when three or four or more situations occur at the same time with each situation interacting with the others to make the point more complex. Without extremely careful consideration by the developer, the monitor may take an erroneous path out of the situation and cause some violation to be detected several hundred instructions later. In cases such as this (which occur very seldom) the system may not be able to recover at all and must stop all jobs.

1.4.2 Unexpected Combinations

Indeed, some of these interacting situations which are seldom encountered are in fact unforeseeable by the programmer when he is developing the monitor. It is not easy, if possible at all, in most cases to determine whether the real error is a programmer's mistake or a combination of unexpected events. Usually some method of feedback is provided to inform the system developers of the failure so that it may be prevented from happening again if possible. The effect of this type of error can be serious if outside means of protection are not employed as discussed later in this chapter.

1.4.3 Outside Errors

Other monitor errors may occur if another part of the system goes awry without being detected. The most frequent of these errors causes either a user program or the monitor to use bad data, make a wrong decision, or otherwise get itself in trouble. The results are the same.

1.4.4 Monitor Self Checks

Some errors detected by the monitor are the results of checks made by the monitor on its own integrity. Frequently, different sections of a monitor are used to perform some specific function using data supplied when the function is needed. Although the exact data is not known when the section is written, the data can often be described to be within a certain range of values or in a specific format. In most cases like this, the monitor checks these parameters of the data before the numerical values are used. For example, an argument (data) for a sub-routine may always have to be between 1 and 10. If the subroutine always checks the argument to be within range, an earlier undetected error may have caused the value to be 15 and the monitor will detect the error. These forms of checking are sometimes called range checks or consistency checks. The recovery depends on the seriousness of the function. If the function is called to support only one job at a time and is not capable of being retried, only that job would be stopped. However, if the function affects all users or the integrity of the operating system, then the monitor will stop all jobs on the system. This is often called "crashing" either a single job or the entire system.

In some cases the system software may arrive at a point or condition that the programmer did not believe possible but coded for the eventuality anyway. Usually the error detected here is minor and affects no jobs. In this instance only a warning is usually given to the system operator and the monitor continues.

1.4.5 Recovery of Monitor Errors

Most errors detected by software, either user or monitor, are considered more serious than any other errors. These errors usually can not be recovered by restarting the function in progress. For this reason almost all of these errors cause the abnormal termination or crash of at least one job on the system. These errors are detected only by the software without any indication of trouble from the hardware. These errors are serious because software or programs do not usually go bad with age. After the program is initially debugged it does not change or degrade because of heat as hardware does. If an error is later detected by the system, it is considered to be caused by an event or eventuality the programmer did not consider, and usually there is no program or function provided to correctly handle the situation.

1.5 HARDWARE ERRORS

The third type of error detected by the system is caused by the hardware or the machine itself. This is the most frequent type of error and the responsibility for identification and correction of these failures falls directly on field service engineers. The system hardware can age and cause intermittent failures. These failures are not permanent, i.e., the failure may not occur during two sequential attempts at the same operation. For this reason the operating systems of today expend a lot of effort to recover from this type of failure. These hardware errors can be divided into three categories:

1. CPU-instruction and addressing failures

2. Controller and channel failures

3. I/O errors.

Because the system hardware cannot be expected to operate continuously without failure, producers of the hardware include facilities to check the hardware operation. The most frequently used error checking scheme is any one of several types of parity networks although many other schemes are available. Once the hardware has detected an error it may either signal the CPU and system software that an error has occurred or attempt to recover from the error and notify the software if it cannot recover successfully.

1.5.1 CPU

Failures occurring in the CPU and main storage section of the system are perhaps the most difficult to handle correctly. These failures can easily modify either the operating system software or a user program or cause instructions to be incorrectly executed. A failure in an addressing section may cause the system to operate with wrong data or unknowingly modify some other job's program or data. For these reasons CPU errors will ordinarily cause the crash of a job or the entire system regardless of whether a user or the monitor is in control. The most recently developed CPU's have attacked this problem by adding more checking circuits specifically designed to stop the bad effects of an error once it has been detected. For example, if a word (either an instruction or data) sent to the CPU by the memory fails a parity check, the operation in progress is stopped before the bad data is used. In this way, the system localizes the effect of the error and the impact of the failure is reduced. The operating system may crash all jobs as a result but the system's data base (user's data files and programs) will not be affected. In other instances the operating system may be able to retry the failing instruction or memory reference successfully and not have to crash any users at all.

1.5.2 Controllers and Channels

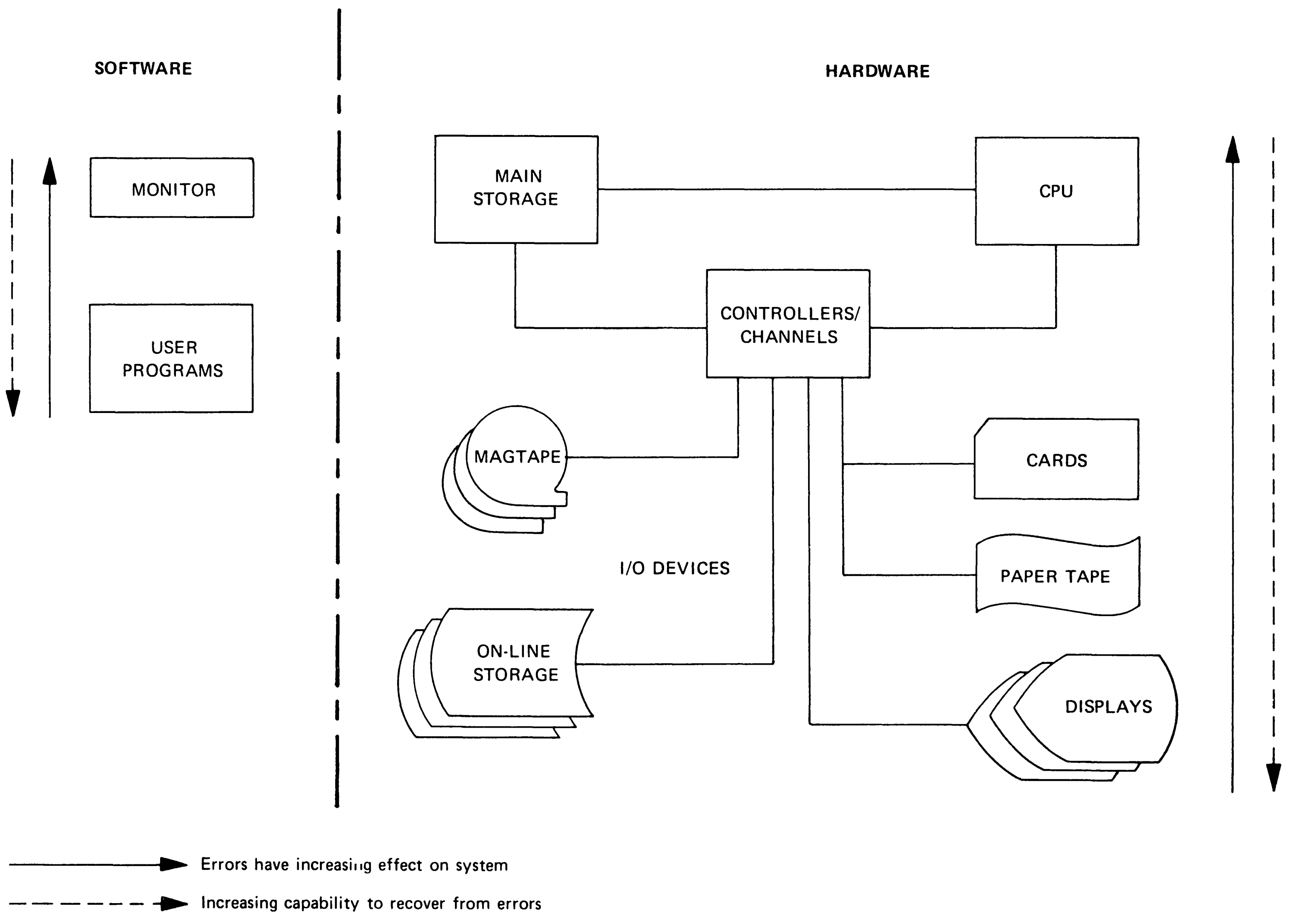
The second major section of the system is that section composed of the various controllers and channels. The system controllers monitor and control several I/O devices of the same type, and the channels of various types connect the CPU and/or main storage units with the I/O controllers or devices. Failures in this portion of the system can usually depend on rather extensive recovery procedures to overcome a problem. However, these errors are likely to affect several jobs or users because each controller or channel can handle several I/O devices being used by many jobs. The checking circuits employed here are of the same type and perform the same function - ensure the device is correctly performing the requested operation and ensure the device is transferring the requested data correctly. Detected errors are signaled to the CPU and monitor and may stop the current operation if the error is serious. An example here might be a controller's parity check of a command issued by the CPU. If this parity check fails, the command would not be performed and the error would be signaled back to the CPU.

The recovery procedure invoked by the operating system may be as simple as retrying the failing operation a number of times or as elaborate as finding another path to the same point, such as using another controller attached to the same group of devices. Some of the controller/channel errors are concerned with data errors. Here the recovery procedure may include correcting the data after it is in main storage using error information provided by the controller or channel.

1.5.3 I/O Errors

Errors detected by a single I/O device are recovered in the same manner as channel or controller failures but usually the error will affect only one job or task. The most frequently used form of error recovery is the simple retrying of the failing operation. If the failure continues for a specified number of consecutive retries, the job or task is crashed. These retry procedures may include other steps every so often during the recovery operation. These steps may include such action as repositioning the heads of a disk drive before every 5th retry or moving a magnetic tape over a tape cleaner mechanism before every 4th attempt to recover. Other forms of I/O error recovery may include moving the data media to a different unit if possible. For example, a reel of magnetic tape (the media) may be moved to a different tape drive and the operation started again.

So far we have seen that there are several methods which may be employed to attempt to recover from errors after they have been detected. Those detected by software alone are more difficult to recover and have more severe impact on the system. Those errors detected by the system hardware vary in both system impact and recoverability. The impact and recoverability of the errors is basically related to the logical distance away from the CPU as summarized in Figure 1-1.


SOFTWARE
HARDWARE
MONITOR
USER PROGRAMS
MAIN STORAGE
CPU
CONTROLLERS/ CHANNELS
MAGTAPE
CARDS
I/O DEVICES
PAPER TAPE
ON-LINE STORAGE
DISPLAYS
Errors have increasing effect on system
Increasing capability to recover from errors

The methods already described only cover the initial attempts made to recover from the error. Many of the errors still result in the abnormal termination or crash of a job or the whole system. Recent hardware and software development in the area of error detection and recovery has reduced the number of errors which result in crashing the entire system. This helps to achieve the goal of localizing the impact of errors. Additionally, new hardware and software have been designed to be more reliable and to fail less often. This all has the effect of increasing system availability which is a measure of a system's continuing ability to handle requests for computation.

1.6 CHECKPOINT/RESTART AND BACKUP

Another aspect of error recovery is the effort involved to get back to the point of processing the job just before the error occurred. Consider, for example, a job that requires eight hours to process. If an error occurs during the 6th hour of the job's run, one of two events will occur; either the error will recover successfully and the job will continue (possibly without even knowledge of the error) or the job will be crashed while other jobs continue to run. If the job was crashed, the recovery cycle would not be complete until the job was back at the point six hours into the "rerun". The recovery time would include the six hours rerun time plus any additional time needed to recover the original data.

In order to reduce this rerun time and help increase availability, features generally known as checkpoint/restart are included in most systems. This technique is simply the stopping of a job at regular intervals and saving in auxiliary storage the current state of the job and any program data, then continuing processing of the job. If a fatal error occurs, the subsequent rerun of the job may start at the last checkpoint instead of at the beginning. In our example, if checkpoints were taken every 1/2 hour, the rerun time would be no longer than 1/2 hour instead of six hours or more. The advantage of this facility is obvious and is always employed in any system environment where jobs are processed on a tight schedule or the output must meet a deadline. The disadvantage is the requirement of the additional auxiliary storage needed to hold the checkpoint.

This same general procedure is also employed to backup the entire system data set. In most computer systems all of the system's data base, both programs and data files, is saved on magnetic tape and stored in special areas such as vaults. By using this facility the system's data is never totally lost in case of a major disaster such as a fire in the computer center. This method of backing up the data may be done monthly, weekly, or even daily depending on the consequence of losing the most current data. Some computer centers may even back up their data to fire storage each time the data is changed.

1.7 OPERATOR MESSAGES AND SYSTEM RECONFIGURATION

The second goal of error detection and recovery is to localize the effects of every detected error. This can also be accomplished by reducing the number of times the error occurs. If the error can be prevented from repeating itself, the effects are limited. In most cases of monitor and hardware errors a message is sent to the operator's teletypewriter or display. Also, special programs have been developed to report the status of the system, including error counts, etc., to the system operator. In this manner the system operator is knowledgeable of what the system is currently doing. If errors start to occur, the operator may seek assistance to determine the cause and find a solution for the errors.

If several errors can be traced to a single unit, the operator may attempt to recover whatever data was lost and then switch the units to a duplicate device and inform the monitor that the defective piece of hardware or software is no longer available. This process is called reconfiguration.

After reconfiguration the system may operate more slowly or at a reduced efficiency rate, but at least still operate until the faulty device can be fixed. In some cases backup units may be used to keep the system running at the same level. This method is rather expensive in terms of additional hardware but may be required in critical applications.

Several of the error messages for the operation will also include directions for corrective action or steps to be performed as part of the recovery sequence. For example, if a deck of cards being read by the system has an error, the message to the operator would state the error and tell the operator to put the deck back into the input hopper of the card reader and restart the job. In another case the operator might be notified of several non-recoverable errors while reading a magnetic tape and the operator may be asked to move the tape to a different unit and try the job again.

More sophisticated operating systems may even mark a device or unit unavailable to itself after the error rate has crossed a specified threshold. In this case, the operator would be notified after the fact and may even be directed to contact the system maintainers about the faulty device.

All of the methods and procedures discussed so far have dealt with detecting errors and controlling the effects of these errors. Any computer system which incorporates all or several of these functions will provide more data integrity for its users. When errors occur, the operating system and hardware will detect them and either attempt to recover or crash the appropriate job to prevent damage to the user, his data, or other jobs on the system. If several errors occur in a non-critical section of the system, the faulty device may be taken out of the system configuration until it can be repaired.

1.8 ERROR REPORTING TO FIELD SERVICE

In addition to providing more data integrity, more recent operating systems help field service repair faulty devices or systems by acting as a form of diagnostic when errors occur. This aspect of an operating system or monitor is called error recording and reporting. This capability has proven itself to be one of the most valuable tools available to field service engineers. This facility has eliminated many hours from system or device repair times, making the field service engineer's job easier and increasing system availability to the customer at the same time.

For the purpose of this discussion, a diagnostic may be considered to consist of only two basic sections. The first section is an exerciser which creates activity (perhaps of a closely controlled type) on some portion, if not all, of a system. Once an error is detected, the second section of the diagnostic generates and presents to its user information concerning the failure. This information either directly identifies the failing component or provides enough information for the user of the diagnostic to determine the failing component. This information is presented to the user, usually field service, in a manner and form that is easily understood by him.

Because a monitor drives all of the system hardware in an interactive manner for long periods of time, it can be considered one of the best exercisers available. Once an error occurs, the error recording sections of the monitor gather all of the available hardware and software information concerning the error and preserve this information in auxiliary storage for later reporting to field service. The reporting section of the package, upon command, presents this information to field service in a manner that is understandable and useful in identifying the failing section or component of the system.

By using this capability long-term, exhaustive diagnostics do not have to be run to recreate the errors and provide error information after the error was originally detected by the monitor. Field service engineers need only collect and analyze this data preserved by the operating system to determine which devices are detecting errors. Using the detailed information concerning these devices, field service can then determine the most efficient method to accomplish the repair, usually not requiring any diagnostic runs at all.

In addition to preserving information regarding hardware and software errors, the monitor may also use this method to save information regarding significant operational events such as system reloads and system activity rates to help in determining overall system performance and error rates for system devices.

This tool for field service, built into operating systems, coupled with a functional level of understanding monitor error detection and recovery procedures, can enable field service engineers to effectively maintain systems in a professional manner with minimum interference to customer's operating schedules.

1.9 SUMMARY

As the usefulness and complexity of computer systems increased with development, so did the dependency on the computer's output. This dependency was on both turnaround time and accuracy of information which are affected by the error detection, recovery, and reporting capabilities of the computer system.

The errors possible from a system are basically:

1. User programming errors

2. Monitor errors

3. Hardware errors

Those errors occurring in either a user's program or the monitor usually have a more serious effect on either the user's job or the entire operating system. Such errors are difficult to recover from because of their complexity, but recent developments have helped to reduce significantly the overall effects of these errors.

Hardware detected errors vary in recovery capabilities depending on where the errors are detected. Recovery procedures may vary from simple retry to more elaborate alternate path methods.

The effects of non-recoverable errors have been reduced through the use of checkpoint/restart and backup procedures. Reoccurring errors may sometimes be prevented by system reconfiguration techniques or backup devices.

All errors are usually reported to the operator of the system and detailed information about errors may be preserved in auxiliary storage for field service analysis. Effective use of these recording facilities of the monitor can enable field service to diagnose system malfunctions without running long duration diagnostics to recreate the problem. This tool may be the most used and most helpful tool in the field service engineer's toolbox.

CHAPTER 2

TOPS20 ERROR DETECTION AND RECOVERY

(to be supplied)

CHAPTER 3

HOW TO RUN SYSERR

3.1 INTRODUCTION

SYSERR is a user program to list the contents of the system error file. To run the program you must be "logged in" and have maintenance privileges. If you are not familiar with how to "log in" to the system, refer to the manual GETTING STARTED WITH TOPS20.

3.2 BEFORE RUNNING SYSERR

After logging in and before running SYSERR you must have two special areas defined for you to access. These two areas are <SYSTEM>, where the error file exists; and <SUBSYS>, where the compatibility package exists. To do this, type the following on your TTY:

```
     @DEFINE(SPACE)SYS:(SPACE)<SYSTEM>,<SUBSYS)(CR)
```

NOTE

All commands which are input to the system are underlined, "(CR)" means carriage return and "(SPACE)" means a single space.

If you have already defined the logical name SYS, redefine it to include <SYSTEM> and <SUBSYS>.

To check your logical name-assignment, type

```
     @INF(SPACE)LOG(CR)
```

NOTE

As described in Appendix C, the SYSERR package consists of 3 modules, SYSERR, SYSERD, and SYSERS. All 3 must reside in the same directory. Normally this directory is <SUBSYS>; however, if the 3 modules are in your own directory the logical name SYS must be defined to include "DSK:" as the first logical name in the assignment for the package to work correctly. The correct command for this is

```
          @DEFINE(SPACE)SYS:(SPACE)DSK:,<SYSTEM>,<SUBSYS>(CR)
```

To call the program SYSERR, type on your TTY:

```
     @SYSERR (CR)
```

and the program will respond with

```
     *
```

indicating it is ready to accept your commands.

3.3 GENERAL COMMAND STRING

The general form of a command string to SYSERR is:

```
     * ODEV:OFILE.TYP=IDEV:IFILE.TYP/SWITCH/SWITCH...(CR)
```

where:

ODEV:	The output device where you want the listing file. May be any device which can perform output.

NOTE

If "ODEV:" is "LPT:" you will get automatic spooling; if "ODEV:" is "PLPT0:" output will go to physical line printer 0.

OFILE.TYP	The name and type of the listing file.
IDEV:	The input device where the system error file resides. May be any device which can perform input.
IFILE.TYP	The name and type of the input file.
/SWITCH:	The control switches which tell SYSERR what types of errors or listings you desire.

It is not necessary to type a full command to SYSERR because certain portions have a default value which SYSERR uses if you have not specified that portion of the command string. The default values used by SYSERR are:

COMMAND PORTION	DEFAULT
ODEV:	DSK: in your own area
OFILE.TYP	The default is the listing control switch specified such as /MASALL. Output file would be MASALL.LST.
IDEV:	SYS:
IFILE.TYP	ERROR.SYS
/SWITCH	/ALLSUM - If this default is used the output file name default is ERROR.LST.

The listing control switches available for use include:

/ALL	LIST ALL ENTRIES
/ALLPAR	LIST ALL THOSE CAUSED BY PARITY ERRORS
/ALLPER	LIST ALL PERFORMANCE ENTRIES
/ALLSUM	GIVE ALL DEVICE SUMMARY
/CPUALL	LIST ALL PROCESSOR RELATED ENTRIES
/CPUPAR	LIST THOSE CAUSED BY PARITY ERRORS
/CPUPER	LIST ALL CPU PERFORMANCE ENTRIES
/CPUSUM	GIVE PROCESSOR SUMMARY
/MASALL	LIST ALL ENTRIES CONCERNING MASSBUS DEVICES (TU16, TU45 & RP04)
/MASPAR	LIST ONLY THOSE CAUSED BY PARITY ERRORS
/MASNXM	LIST THOSE CAUSED BY NXM
/MASSUM	LIST SUMMARY INFORMATION

3.4 OTHER CONTROL SWITCHES

Other control switches are also available to further control the listing. This type of switch is used to select a particular device, group of devices, or only errors occurring during a specific date/time period. Switches of this type include:

/BEGIN:MM-DD-YY:HH:MM:SS	Begin listing of entries logged on date specified by MM-DD-YY. Other date formats such as DD-MM-YY and JAN-16-1976 are acceptable.
/END:MM-DD-YY:HH:MM:SS	End listing of entries on the date specified. The same formats are acceptable.
/DEV:name /DEV:type	Select for listing only those entries which involve the device specified by name or type. Available device types include KLCPU, 11CPU, LP20, CD20, DH11, TU45, TU16, and RP04. To indicate a specific disk drive (DP) or magtape drive (MT) by /DEV:name, you must use the form DPabc or MTabc, where

a = the logical controller address.

b = the logical MASSBUS address.

c = the logical slave address for MT and 0 for DP.

NOTE

You will find these logical addresses, by generating the first summary listing.

If /DEV:name is used, the listing control switch, such as /MASALL, must be used.

For TOPS20 systems, using only /DEV:type listed above without a listing switch, such as /MASALL, causes SYSERR to examine each entry and force listings for those entries whose device type match that specified.

/DETAIL:	List all information for Massbus and magtape instead of brief listing. May be abbreviated to "/DET".
/RETRY:	List only those entries whose retry count is greater than the value specified.

3.5 EXAMPLES

Following are several examples of command strings and explanations of how they are interpreted by SYSERR.

EXAMPLE #1

```
     * TTY: = (CR)
```

This is the first command which should be given to SYSERR. It will list summary information about the entire contents of the error file on your TTY. By examining this printout you may determine those portions of the system which are of interest to you and give further commands to SYSERR to list only the desired reports. Note that this command used several default values. The values which were defaulted are enclosed in [ ] and if the whole command were typed it would look like:

```
     *TTY:  [ERROR.LST]=[SYS:ERROR.SYS/ALLSUM] (CR)
```

EXAMPLE #2

```
     * TTY:  =/BEGIN:-1D (CR)
```

This is basically the same command as Example #1. However, this time only those errors which have occurred in the last 24 hours are considered. The value specified in the /BEGIN: switch may be as shown, or changed to increase number of days (-7D for 1 week), or a specific date included as described under /BEGIN:

EXAMPLE #3

```
     * =/DEV:RP04/DETAIL (CR)
```

This command tells SYSERR to provide complete, detailed reports for all entries which concerned any RP04 in an output file called ERROR.LST on your disk area. Again several defaults were used.

EXAMPLE #4

```
     * = (CR)
```

This is the easiest command to type to SYSERR and it uses all of the default values. It is identical to the action of Example #1 except that the listing file, called ERROR.LST is generated on your disk area.

EXAMPLE #5

```
     * =/MASALL/BEGIN:JAN-1-76:13:00/END:JAN-7-76:13:00/DEV:DP030/DETAIL(CR)
```

This command will tell SYSERR to create a file named MASALL.LST on your disk area which contains detailed information about all the errors detected by device DP030 between the period from 1 PM on Jan. 1, 1976 to 1 PM Jan. 7, 1976.

EXAMPLE #6

```
     * MTA1: =MTA2:/MASALL/CPUALL/BEGIN:-30D/END:-3D (CR)
```

This command tells SYSERR to create the list file on MTA1 and read the error file from MTA2. All entries concerning either a massbus device or the CPU during a period beginning 30 days ago and ending 3 days ago will be listed. This example points out that SYSERR can process multiple commands from one command string and does not need to always have the input and output files on disk.

EXAMPLE #7

```
     * /HELP (CR)
```

This command tells SYSERR you have forgotten how to give SYSERR commands. The program will list a HELP file on your TTY which gives abbreviated information on how to run the SYSERR program.

After each command is processed by SYSERR, the program gives a carriage return, line feed and another prompt character (*) indicating it is ready for another command. If no more commands are required, type

```
     * ↑C  (control C)
```

and the program will exit back to monitor. If the output files were created on your disk area they may be listed on the line printer with the monitor command

```
     @ PRINT XXX.LST (CR)
```

where XXX is the name of the file you want to list.

3.6 OTHER COMMANDS

If you are running SYSERR on an LA36 with wide paper, such as from the local DEC office, an additional command to the monitor before you call SYSERR will allow the full width of the paper to be used when summary listings are printed on this TTY. The command is:

```
@ TERMINAL WIDTH 132 (CR)
```

3.7 INDIRECT COMMANDS

SYSERR has the capability of processing commands from a disk file as well as from your TTY. This is called indirect command files and is useful if you have several "favorite" commands to use in succession. To use this function create a file of commands just as you would type them on your TTY.

NOTE

SYSERR does not support line-sequence numbers.

To tell SYSERR to use this file the command is:

```
* @ DEV:NAME.TYP (CR)
```

where DEV: is the location of the file (DEFAULT is DSK:) and NAME.TYP is the name of the command file.

CHAPTER 4

SYSERR REPORT FORMATS

4.1 INTRODUCTION

This chapter describes each of the reports generated by SYSERR. It is the intent of SYSERR to make each report self-explanatory for those people who are knowledgeable of the system. This chapter is included to provide information for those who are not familiar with the system or who are inexperienced with SYSERR.

4.2 REPORTING CONVENTIONS USED IN SYSERR

All numbers output by SYSERR are either octal, decimal, or otherwise noted. All decimal values are followed with a period (.) to indicate that they are decimal. All other values are octal. Values printed in half-word format have leading zeros suppressed in each half of the word and the halves are separated with a comma (,).

All register values which are translated to text, such as a CONI value, have text translations only for bits or bytes of interest and the whole value is dumped. For example, the CONI value listed might include a DONE bit and a PI assignment, but these bits are not translated to text.

All dates and times used by SYSERR, both in command strings and report listings are local time unless otherwise stated. The internal day/time maintained by the TOPS20 monitor and all day/time values stored in the error file are recorded as GMT.

4.3 HEADER FORMAT

The top portion of each report is the header. It describes the entry type, when the entry was recorded by the monitor, the monitor run time, or uptime, at the time the entry was recorded, and the serial number of the CPU where the error was detected.

```
***************************************************
TOPS20 SYSTEM RELOADED(CODE 101)
 LOGGED ON MON 12 JAN 76  9:35:05PM      MONITOR UPTIME WAS  0:00:32
        DETECTED ON SYSTEM # 1031.
***************************************************
```

The code number in parenthesis after the report name is the event type number, as described in Appendix B, and is used by SYSERR to determine how to list this entry.

4.4 TOPS20 SYSTEM RELOAD

SAMPLE:

```
***************************************************
TOPS20 SYSTEM RELOADED(CODE 101)
 LOGGED ON TUE 13 JAN 76  6:15:03PM      MONITOR UPTIME WAS  0:00:22
        DETECTED ON SYSTEM # 1031.
***************************************************

CONFIGURATION INFORMATION
        SYSTEM NAME:     V 1.02.35, TOPS-20 DEVELOPMENT SYSTEM #1031
        MONITOR BUILT ON:        FRI  9 JAN 76  7:36:27PM
        CPU SERIAL #:            1031.
        MONITOR VERSION:         10235.
RELOAD BREAKDOWN:
        WHY RELOAD:      SA
```

This entry is created each time the TOPS20 monitor is loaded. The configuration information section includes the system name specified at the time the monitor was built, the version number and the date the monitor was built.

4.4.1 Reload Breakdown

This section explains why the monitor was reloaded. If a BUGHLT occurred and the system was set for auto-reload, the BUGHLT address will be listed and a Code 102 (BUGHLT/BUGCHK Report) entry will provide information about the BUGHLT which caused the reload. If the reload was other than an auto-reload caused by a BUGHLT, this section will list the operator's answer to the "WHY RELOAD" question asked by the system software at startup. There are no restrictions on what the operator may say; however, the answer should describe either what happened to cause the reload, such as "BAD MICROCODE" or the expected future status of the system such as "NEW VERSION," or "SCHEDULED."

4.5 TOP20 BUGHLT-BUGCHK

SAMPLE:

```
***************************************************
TOPS20 BUGHLT-BUGCHK(CODE 102)
 LOGGED ON TUE 13 JAN 76  1:35:16PM      MONITOR UPTIME WAS  2:36:29
       DETECTED ON SYSTEM # 1031.
***************************************************

       SYSTEM NAME:    V 1.02.35, TOPS-20 DEVELOPMENT SYSTEM #1031
       SYSTEM SERIAL #:        1031.
       MONITOR BUILT ON:       FRI  9 JAN 76  7:36:27PM
       MONITOR VERSION:        10235.

ERROR INFORMATION:
       DATE-TIME OF ERROR:     TUE 13 JAN 76  1:35:10PM
       # OF ERRORS SINCE RELOAD:       1.
       FORK # & JOB #:         51,12
       USER'S  CONNECTED DIR,
         LOGGED IN DIR:TPORADA , MCKIE
       PROGRAM NAME:           EXEC
       ERROR:          BUGCHK
       ADDRESS OF ERROR:       52211
       NAME:                   ILLUUO
       DESCRIPTION:            KIBADU: ILLEGAL UUO FROM MONITOR CONTEXT
       CONTENTS OF AC'S:
    0:  0,0
    1:  0,215365
    2:  0,303770
    3:  0,30
    4:  777777,13
    5:  40000,0
    6:  0,100000
    7:  0,51
   10:  3,0
   11:  0,777777
   12:  0,370
   13:  22,356774
   14:  0,0
   15:  260740,301107
   16:  0,0
   17:  777642,777541
       PI STATUS:       0,177
       SELECTED VALUES:        2
       0,0
       4000,1
       0,0
       0,0
```

This report is generated each time the TOPS20 monitor detects any one of three general types of monitor software errors: BUGHLT, BUGCHK, or BUGINF. The most serious of these is BUGHLT which will always crash the system. At this point something is very seriously wrong and the monitor doesn't have enough integrity to attempt any further error recovery. The monitor will, however, collect pertinent information for error recording. When the monitor is reloaded, this information will be extracted from the crash dump file, if present, and transferred to ERROR.SYS. BUGCHK and BUGINF are less serious, perhaps correctable monitor-detected errors which may only affect particular users instead of the entire system. These errors may or may not crash the user depending on the error which occurred. For a more complete description of these types of errors, refer to Chapter 2.

4.5.1 Report Contents

The upper section of this report describes the version and name of the running monitor and is identical to the same section of the system reload report. The ERROR INFORMATION section contains the majority of information for this error. The date and time of the error are included primarily to cover the situation of a BUGHLT finally being reported some length of time after it occurred.

The number of errors since reload are listed because only 5 occurrences of this type error entry are allowed in the monitor's error recording buffer at any one time. In the case of an error occurring in a tight loop, more than 5 entries could overflow the buffer and the information for the first (and usually most interesting) occurrence might be lost. These numbers should increment by one for each report listing; however, if the sequence is broken, it is an indication that more than 5 entries occurred before the error logger module in the monitor could empty the buffer.

The FORK # and JOB # are the numbers associated with the current user at the time of the error. A value of -1 or 777777 indicates that the monitor was performing an overhead function (such as scheduling) and there was no current user. Note that the FORK # and JOB # indicate the current user and not necessarily the user being serviced by the monitor interrupt level routines (e.g., BUGCHK detected at interrupt level during I/O for a different user).

The user's connected directory and logged in directory are also for the current user and are listed along with the user's program name to aid in identifying the person running at the time of the error. If several reports indicate the same user and/or program, talking with that user or examining that program should help in identifying the source of the problem.

Following the user identification is information specifically identifying the name and description of the error. If the "/DETAIL" switch is used with the SYSERR command string, more information will be listed which is useful for further analysis of the error. Included are the contents of the monitor's block of AC's and the PI system status. Some particular errors will also include "SELECTED VALUES." A maximum of 4 values may be preserved in the error file. Description of these values is dependent on the type of error which occurred and may be obtained from the monitor listings.

4.6 MASSBUS DEVICE ERRORS

This entry is recorded in the ERROR.SYS file by the monitor each time an error is detected in the Massbus System including the Massbus devices (RP04, TU45, and TU16), the RH20 controller, and certain errors occurring in the channel logic.

4.6.1 Sample Report

The next two pages show sample Massbus device error reports.

```
***********************************************
MASSBUS DEVICE ERROR(CODE 111)
 LOGGED ON TUE 13 JAN 76 12:15:03PM      MONITOR UPTIME WAS  1:16:17
        DETECTED ON SYSTEM # 1031.
***********************************************

        UNIT NAME:      DP110
        UNIT TYPE:      RP04
        VOLUME ID:
        LBN:            59184.  =
        CYL:   155.     SURF:   14.     SECT:   4.
        OPERATION AT ERROR:      DEV.AVAIL., GO +  READ DATA(70)
        USER'S CONNECTED DIR,
          LOGGED IN DIR:UNKNOWN , UNKNOWN
        USER'S PGM:
        USER'S FILE:
        FINAL ERROR STATUS:      200000,7
        RETRIES PERFORMED:       2.
        ERROR:  RECOVERABLE DRIVE EXCEPTION,
CONTROLLER INFORMATION:
        CONTROLLER:     RH20 # 1
        CONI AT ERROR:  0,202415 = DRIVE EXCEPTION,
        CONI AT END:    0,2405 =  NO ERROR BITS DETECTED
        DATAI PTCR AT ERROR:     732201,177471
        DATAI PTCR AT END:       732201,177771
        DATAI PBAR AT ERROR:     720001,7004
        DATAI PBAR AT END:       720001,7007
CHANNEL INFORMATION:
        CHAN STATUS WD 0:        200000,200374
           CW1:  620000,731000  CW2:  0,0
        CHN STATUS WD 1:         540100,200375 = NOT SBUS ERR,NOT WC = 0,LONG WC ERR,
        CHN STATUS WD 2:         604000,731600
DEVICE REGISTER INFORMATION:
                 AT ERROR        AT END          DIFF.           TEXT
        CR(00): 4070            4010            60               DEV.AVAIL., READ DATA(70)
        SR(01): 50700           10700           40000           ERR,MOL,DPR,DRY,VV,
        ER(02): 100000             0            100000          DCK,
        MR(03): 400             400                0
        DA(05): 7007            7010            17              D. TRK = 16, D.SECT. = 7
        DT(06): 24020           24020              0
        LA(07): 1740            1320            460
        OF(11): 100000          100000             0            AT END: OFFSET = NONE
        DC(12): 233             233                0            155.
        CC(13): 233             233                0            155.
        EP(16): 5432               0            5432
        PL(17): 2000               0            2000

DEVICE STATISTICS AT TIME OF ERROR:
# OF READS:      94212.  # OF WRITES:    87776.  # OF SEEKS:      20330.
# SOFT READ ERRORS:      1.      # SOFT WRITE ERRORS:    0.
# HARD READ ERRORS:     0.       # HARD WRITE ERRORS:    0.
# SOFT POSITIONING ERRORS:       0.      # HARD POSITIONING ERRORS:       0.
# OF MPE:  0.    # OF NXM:  0.   # OF OVERRUNS:  0.
```

```
**********************************************
MASSBUS DEVICE ERROR(CODE 111)
 LOGGED ON TUE 13 JAN 76  1:41:18PM       MONITOR UPTIME WAS  2:42:31
       DETECTED ON SYSTEM # 1031.
**********************************************

       UNIT NAME:      MT052
       UNIT TYPE:      TU45
       UNIT SERIAL #:  0024
       VOLUME ID:
       LOCATION:    RECORD  #  664.    OF FILE #  0.
       OPERATION AT ERROR:      DEV.AVAIL.  GO +  READ FWD.(70)
       USER'S CONNECTED DIR,
         LOGGED IN DIR:DEMO-1 , DEMO-1
       USER'S PGM:     AEGIS
       USER'S FILE:
       FINAL ERROR STATUS:     0,1
       RETRIES PERFORMED:      31.
       ERROR:  NON-RECOVERABLE DRIVE EXCEPTION,
CONTROLLER INFORMATION:
       CONTROLLER:     RH20 # 0  TM02 #:  5
       CONI AT ERROR:  0,202415 = DRIVE EXCEPTION,
       CONI AT END:    0,202415 = DRIVE EXCEPTION,
       DATAI PTCR AT ERROR:    732205,177771
       DATAI PTCR AT END:      732205,177771
       DATAI PBAR AT ERROR:    720005,0
       DATAI PBAR AT END:      720005,0
CHANNEL INFORMATION:
       CHAN STATUS WD 0:       200000,43572
         CW1:  400320,546763  CW2:  615520,257000
       CHN STATUS WD 1:        500000,43574 = NOT SBUS ERR,
       CHN STATUS WD 2:        600000,257665
DEVICE REGISTER INFORMATION:
               AT ERROR        AT END          DIFF.           TEXT
       CR(00): 4070            4070               0             DEV.AVAIL.  READ FWD.(70)
       SR(01): 54620           54620              0            ERR,MOL,WRL,DPR,DRY,SDWN,
       ER(02): 100300          100300             0            COR/CRC,PEF/LRC,INC/VPE,
       MR(03): 45600           45600              0
       FC(05): 3410            3410               0
       DT(06): 142012          142012             0
       CK(07): 774             774                0
       SN(10): 44              44                 0
       TC(11): 101062          101062             0            ACCL, 800BPI NRZ 10 COMPATIBLE SLAVE #2

DEVICE STATISTICS AT TIME OF ERROR:
# OF READS:     2754.   # OF WRITES:    0.      # OF SEEKS:     0.
# SOFT READ ERRORS:     4.      # SOFT WRITE ERRORS:    0.
# HARD READ ERRORS:     8.      # HARD WRITE ERRORS:    0.
# SOFT POSITIONING ERRORS:      0.      # HARD POSITIONING ERRORS:      0.
# OF MPE:  0.   # OF NXM:  0.   # OF OVERRUNS:  0.
```

4.6.2 Report Description

The UNIT NAME refers to the physical Massbus unit active at the time of the error. This is a 5 character name of the format XXABC where:

XX is the device type

DP = disk drive (RP04)

MT = mag tape (TU45 and TU16)

A is the logical address of the RH20 controller for this device (0-7)

B is the logical Massbus address for this device (0-7). For magtape units this is the TM02 address on the Massbus

C is the slave number of this magtape unit. For RP04 devices this number is always 0.

The LBN listed for RP04 reports refers to the Logical Block Number of the pack being addressed when the error occurred and is translated to cylinder, surface, and sector to provide physical location. For magtapes, the record and file number are listed to show location.

The OPERATION AT ERROR is a decode of the last command issued to the device before the error occurred. If this command does not agree with the listed contents of the device's control register (with the exception of the GO bit) an error in the control bus may have occurred. The user's connected and logged in directory, and program are listed for magtape units to aid in finding bad tapes which may be causing errors. The ERROR statement is a text translation of the RH20 CONI at error. If the error was non-recoverable, the monitor sets bit 2 in the IORB status word (MB%IRS) and SYSERR states the error is NON-RECOVERABLE.

The CONTROLLER INFORMATION lists the controller type and logical address; and (for magtapes only) the TM02 logical address. This section also lists the CONI's and DATAI's. PTCR is Primary Transfer Control Register and PBAR is Primary Block Address Register. The CHANNEL INFORMATION lists the contents of the channel's status and logout area. The values listed for CW1 and CW2 are the contents of the address pointed to in the right half of Channel Status Word 0 and the contents of the address pointed to +1.

The DEVICE REGISTER INFORMATION lists the contents of the device's registers at the time of error and after the last retry, the XOR difference and the text translation for the value at error. The only exception to this is the RP04 OFFSET register. The text translation for this register is the value at end and is noted by "AT END:".

If both the AT ERROR and AT END values for any register are zero, that register is not listed.

The DEVICE STATISTICS provide an indication of the error rate. The number of reads and writes for magtape indicate frames of tape transferred and for disks this is the number of blocks transferred.

4.7 FRONT END DEVICE ERRORS

These types of entries are recorded in the system error file by the monitor as a result of a request for error logging from the front end. These errors are detected by the front end and it gathers the error information and passes the packet to the monitor across the DTE-20 for error logging. The errors detected by the front end fall into two basic types: those concerning the front end hardware and software; and those concerning the KL CPU hardware and software. Descriptions of the error detection, recovery and reporting for the front end may be found in Chapter 2. Currently, reports are created for the following "devices:" - LP20, CD20, DH11, KLCPU, and KLERROR.

4.7.1 Report Description

The top section for all reports is basically the same and includes the DTE-20 logical address for this front end, the version number of the front end Software, the FORK # and JOB # associated with this error. If the FORK # and JOB # are 777777, 777777, this is an indication that the TOPS20 monitor knows of this device but it is not currently assigned to any fork or job. 777776, 777776 indicates the TOPS20 doesn't know anything about this device.

The upper section of these reports also includes the user's connected and logged in directories and program name as well as the device name and logical address. It also lists the octal value and text translation for the standard status word generated by the front end for each transfer across the DTE-20. It is the ERROR LOG REQUEST bit in this word which causes the packet to be recorded into the error file.

The remainder of each device report is dependent on the type of device being reported. If SYSERR does not know how to list a device, this fact will be stated in the report and the entry will be listed in octal.

4.7.2 LP20 Report Description

```
************************************************
FRONT END DEVICE ERROR(CODE 130)
 LOGGED ON WED  7 JAN 76 12:38:20PM      MONITOR UPTIME WAS  0:21:45
       DETECTED ON SYSTEM # 1031.
************************************************

       DTE20 #:               0.
       FE SOFTWARE VER:       0.
       FORK #,,JOB#:   35,15
       USER'S CONNECTED DIR,
         LOGGED IN DIR:PORCHER , PORCHER
       USER'S PROGRAM: TECO
       DEVICE: LP20  # 0.

       STD. STATUS:     120 = ERROR LOG REQUEST,
       LP20 GEN STATUS:         0 =  NO ERROR BITS DETECTED
       LP20 DEVICE REGISTERS
       LPCSRA: 114102   = ERROR,DAVFU,ON LINE,INT ENB,PAR ENB,
       LPCSRB: 10003    = LPT DATA PAR,DEMAND TIMEOUT,GO ERROR,
       LPBSAD: 154504
       LPBCTR: 354
       LPPCTR: 7734
       LPRAMD: 10000
       LPCCTR: 152     COL. CNTR. = 0 CHAR. BUF = 152

       LPTDAT: 155412  CHKSUM = 333 LPT DATA = 12
```

The device-specific section of this report includes the LP20 GENERAL STATUS created by the front end software and contents of the various LP20 controller registers. Text translations are also included for both status registers and LPCCTR and LPTDAT. If the contents of a register are zero in the error file, that register if not listed.

4.7.3 CD20 Report Description

```
************************************************
FRONT END DEVICE ERROR(CODE 130)
 LOGGED ON THU  8 JAN 76 11:08:49AM      MONITOR UPTIME WAS  2:02:50
       DETECTED ON SYSTEM # 1031.
************************************************

       DTE20 #:               0.
       FE SOFTWARE VER:       0.
       FORK #,,JOB#:   777777,777777
       USER'S CONNECTED DIR,
         LOGGED IN DIR:UNKNOWN , UNKNOWN
       USER'S PROGRAM:
       DEVICE: CD20  # 0.

       STD. STATUS:     106 = ERROR LOG REQUEST,HDWR ERR OPR REQ'D,OFF LINE,
       CD20 GEN STATUS:         17       = HOPPER EMPTY,STACK CHECK,PICK CHECK,READ CHECK,
       CD20 DEVICE REGISTERS
       CD11ST  10304    = OFF-LINE,READY,INT ENB,HOPPER CHECK,
       CD11DB: 177777
```

As with the LP20 the remainder of this report includes the CD20 GENERAL STATUS word maintained by the front end software and the octal contents and text translation of the CD11 device controller's registers.

4.7.4 DH11

This entry is created by the front end each time it detects one of two errors associated with a DH11. These two errors are DEVICE HUNG and LOST INTERRUPT. Samples of each report are shown below.

```
***********************************************
FRONT END DEVICE ERROR(CODE 130)
 LOGGED ON MON  5 JAN 76 12:52:55AM      MONITOR UPTIME WAS  0:06:06
        DETECTED ON SYSTEM # 1031.
***********************************************

        DTE20 #:               0.
        FE SOFTWARE VER:       0.
        FORK #,,JOB#:    777776,777776
        USER'S CONNECTED DIR,
          LOGGED IN DIR:UNKNOWN , UNKNOWN
        USER'S PROGRAM:
        DEVICE: DH11  # 0.

        STD. STATUS:     1100 = DEV HUNG,ERROR LOG REQUEST,
        CONTENTS OF COUNTERS:   24
        36
        36
        40



***********************************************
FRONT END DEVICE ERROR(CODE 130)
 LOGGED ON MON  5 JAN 76 12:48:58AM      MONITOR UPTIME WAS  0:02:09
        DETECTED ON SYSTEM # 1031.
***********************************************

        DTE20 #:               0.
        FE SOFTWARE VER:       0.
        FORK #,,JOB#:    777776,777776
        USER'S CONNECTED DIR,
          LOGGED IN DIR:UNKNOWN , UNKNOWN
        USER'S PROGRAM:
        DEVICE: DH11  # 0.

        STD. STATUS:     2100 = LOST INTERRUPT,ERROR LOG REQUEST,
        PAGE ADDR OF DH WHICH FAILED:   160020
```

4.7.5 KLCPU

This entry is created each time the front end reloads the CPU without the front end itself reloaded. If both are reloaded the generated entry is SYSTEM RELOADED (CODE 101). If the KL reloads the front end a FRONT END RELOADED (CODE 131) entry is created. If the front end reloads the KL, this report is created. The report includes the reason for reload as determined by the front end software.

```
**********************************************
FRONT END DEVICE ERROR(CODE 130)
 LOGGED ON TUE 13 JAN 76 10:59:45AM      MONITOR UPTIME WAS  0:00:58
        DETECTED ON SYSTEM # 1031.
**********************************************

        DTE20 #:                0.
        FE SOFTWARE VER:        0.
        FORK #,,JOB#:    777776,777776
        USER'S CONNECTED DIR,
          LOGGED IN DIR:UNKNOWN , UNKNOWN
        USER'S PROGRAM:
        DEVICE: KLCPU
        STD. STATUS:     100 = ERROR LOG REQUEST,
        KL RELOAD STATUS FROM FRONT END:  20      = KEEP ALIVE STOPPED,
```

4.7.6 KLERROR

This report is perhaps the most complex of the error file entries. If the KL clock stops for any of several errors (FAST MEMORY PARITY ERROR, CRAM PARITY ERROR, DRAM PARITY ERROR, or FIELD SERVICE STOP) a software routine is called in the front end to gather a snapshot of the KLCPU. This routine creates an output file on either the dual-ported RP04 or the floppy disk. The next time the system is reloaded the front end passes the contents of this file across the DTE20 and then into the system error file in several entries. SYSERR pieces this file back together in core and then lists its contents. For a complete description of this portion of error detection and recording, refer to Chapter 2.

SAMPLE REPORT

The following page shows an example of the report for the first record. The listing format of the 2nd record is identical.

```
***********************************************
FRONT END DEVICE ERROR(CODE 130)
 LOGGED ON TUE 13 JAN 76 10:59:46AM      MONITOR UPTIME WAS  0:00:59
        DETECTED ON SYSTEM # 1031.
***********************************************

        DTE20 #:                0.
        FE SOFTWARE VER:        0.
        FORK #,,JOB#:    777776,777776
        USER'S CONNECTED DIR,
          LOGGED IN DIR:UNKNOWN , UNKNOWN
        USER'S PROGRAM:
        DEVICE: KLCPU
        CONTENTS OF KLERROR FILE:
CONTENTS OF RECORD #1  SEEN AT FIRST ERROR
        CREATED:  1:13:76 AT: 10:13:10  FILE FORMAT VERSION:    1       RECORD LENGTH:  1000 BYTES
        U-CODE VER:     0.      FE SER #:       0.      KL SER #:       0.
        MONITOR VER:    0.      FE SOFTWARE VER:        0.
        ERROR CODE:     0 = NONE
        DTE DIAG STATUS:        2104
        DTE DIAG 1:     1400    DTE DIAG 2:     0       DTE DIAG 3:     21000
VALUES RETURNED FROM DIAGNOSTIC FUNCTION READS
   100: 000377602664   101: 000000002600   102: 000013400222   103: 000024252024   104: 000000032422   105: 000000002421
   106: 000000642000   107: 000000614640   110: 001107067462   111: 001500002003   112: 001100000000   113: 000000000000
   114: 000000005700   115: 000033702157   116: 000000100000   117: 700300005700   120: 000000000000   121: 000000000001
   122: 000000000001   123: 000072000000   124: 070054060000   125: 014060720000   126: 000000414000   127: 130046404000
   130: 320060204000   131: 121210021207   132: 530504010303   133: 000201000001   134: 500000001010   135: 001210021207
   136: 000000042004   137: 000000002004   140: 000000040004   141: 000000000104   142: 000000052405   143: 000000002405
   144: 000000050005   145: 000000000125   146: 211003016024   147: 211006276700   150: 211006206014   151: 000001044000
   152: 000001044014   153: 600001044031   154: 000001044031   155: 100020000204   156: 360000726722   157: 020100635722
   160: 051600035230   161: 200002235322   162: 146010137664   163: 265040125355   164: 020000337365   165: 340000533305
   166: 000000000000   167: 000000000000   170: 013063000000   171: 000000000000   172: 000000000000   173: 000000000000
   174: 000000000000   175: 000000000000   176: 000000000000   177: 000000000000
```

4.7.6.1 Report Description - The date and time in the header of this report indicates when the last packet of the KLERROR file was entered in the system error file. The date and time following CREATED is the date and time of the error as recorded by the front end. The format is MM:DD:YY AT: HH:MM:SS. The RECORD LENGTH is the length of this record in PDP11 bytes (8 bits=1 byte). The ERROR CODE is the octal value and text translation of what error was seen (CRAM PARITY, DRAM PARITY, etc.) when the snapshot was taken. The DTE DIAG words are the values read by the front end from these registers in the DTE-20. The values returned from the diagnostic function reads are identified with the FR number and the values listed are 12 octal characters (36 bits) wide and are in the same format as that listed on the CTY when performing a function read command.

If the front end has problems gathering and recording this snapshot, an error code is included in the file and the SYSERR listing will state that the data may not be valid and list the 3 character error code stored in the error file. These error codes are listed here in alphabetical order with an expanded description of the error:

FRONT END ERROR CODES

Code	Description
APC	Cache Directory Parity Error
AMP	Memory Bus Parity Error
APN	KL10 NXM
APP	KL10 Page Failure
APR	KL10 APR Error
APS	KL10 S-Bus Error
BAE	Argument out of range
BUG	Command not implemented
CAE	KL10 CRAM Address Error
CCC	Cannot Clear KL10 Clock
CCR	Cannot Clear KL10 RUN Flop
CCS	Cannot Start KL10 Clock
CES	KL10 Clock Error Stop
CFH	Cannot Find KL10 Halt Poop
CSC	Cannot Sync KL10 Clock
CSR	Cannot Set KL10 RUN Flop
DAE	KL10 DRAM Address Error
DMF	Deposit KL10 Memory Failed
DNP	DTE-20 Not Privileged
DSF	DTE-20 Status Failure

EPE	E-Bus Parity Error
ECT	EBox Clock Timeout
EOA	PDP11 ODD Address Illegal
ESD	EBox Stopped-Deposit
EMF	Examine KL10 Memory Failed
ESE	EBox Stopped - Examine
FRF	Function Read Failed
FWF	Function Write Failed
FXF	Function XCT Failed
IAS	Illegal Argument for Set/Clear Command
IFC	Illegal Function Code
ILA	Illegal Argument for Command
ILC	Illegal Character In String
ILL	Illegal Line to Command Passer
ILS	Illegal Separator Character "?"
IPE	Internal Program Error
KLN	Program Error - KL10 Number Out of Range
KLR	Illegal while KL10 Running
KNC	KL10 Cannot be Continued
MAE	Examine/Deposit Mode Illegal
MUL	Ambiguous Command
NSC	No Such Command
OFC	Odd Function Code
RPT	Repeat Count Changed
RSX	Impossible Error From RSX
SAZ	Zero Starting Address Illegal
TAA	Task Already Active
TIN	Illegal Name for Task
TNS	No Such Task
UNL	Micro Code Not Loaded
VFY	Verify Cycle Failed
WRM	Command Not Available

XCR	KL10 Did Not Return to Halt Loop
UME	Unmatched Error Code

As discussed in Chapter 2, the KLERROR file may contain 2 records. If both exist in the system error file, both will be listed by SYSERR using identical formats.

4.8 FRONT END RELOADED

```
***********************************************************
FRONT END RELOADED(CODE 131)
 LOGGED ON MON  5 JAN 76  1:22:36PM        MONITOR UPTIME WAS  0:02:55
        DETECTED ON SYSTEM # 1031.
***********************************************************

        FRONT END #:    PRIVILEGED  0
        STATUS AT RELOAD:        NO ERROR BITS DETECTED
        RETRIES:        0
        FILENAME FOR DUMP:      <SYSTEM>0DUMP11.BIN.2, 5-JAN-76 13:22:31
```

This report is recorded in the error file each time the KL CPU detects that the front end has halted or is in a loop. The KL will attempt to copy a crash dump file onto disk from the front end's memory and then reboot the front end.

4.8.1 Report Description

The front end number is the logical address of this front end and states if this FE is privileged. The status at reload describes in text any errors which occurred during the reboot process. The filename of the core dump is also listed if the crash dump was successful. This information will be useful to determine the reason for the front end failure.

4.9 PROCESSOR PARITY TRAP

```
***********************************************
PROCESSOR PARITY TRAP(CODE 160)
 LOGGED ON WED 21 JAN 76  2:06:32AM      MONITOR UPTIME WAS  4:26:56
        DETECTED ON SYSTEM # 1031.
***********************************************
STATUS AT ERROR:
        BAD DATA DETECTED BY:    AR
        PAGE FAIL WD AT TRAP:    366420,252000
        BAD DATA WORD:  123457,654321
        GOOD DATA WORD: 123456,654321
        DIFFERENCE:     1,0
        PHYSICAL MEM ADDR.
         AT FAILURE:    0,2415
        RECOVERY:       CONT. USER
        RETRY COUNT:    2.

        FORK #  & JOB #:                23,14
        USER'S  CONNECTED DIR,
          LOGGED IN DIR:ACCOUNTS , OPERATOR
        PROGRAM NAME:           DDT
```

This report is recorded each time a page fail trap occurs in the CPU as a result of an AR, ARX or PAGE TABLE parity error. The monitor will attempt to recover from these errors as described in Chapter 2 of this manual.

4.9.1 Report Description

The information listed as GOOD DATA WORD is valid only if the error is recoverable, otherwise the data will be 0,0 and the DIFFERENCE DATA will be a copy of the BAD DATA WORD. The DIFFERENCE is the result of an XOR between the bad and good data words. If this error also was detected with an APR interrupt, a Processor Parity Interrupt report will be generated.

4.10 PROCESSOR PARITY INTERRUPT

```
***********************************************
PROCESSOR PARITY INTERRUPT(CODE 161)
 LOGGED ON TUE 13 JAN 76 10:59:46AM      MONITOR UPTIME WAS  0:01:00
        DETECTED ON SYSTEM # 1031.
***********************************************

        CONI APR:       3440,513 = MB PAR ERR,SBUS ADDR PAR ERR,
        ERA:            26000,445614 = WD #0 E-BOX STORE
        BASE PHY. MEM ADDR.
         AT FAILURE:    445614

        PC AT INTERRUPT:        63321
        # ERRORS ON THIS SWEEP  0.
        LOGICAL AND OF
        BAD ADDRESSES:  777777,777777
        LOGICAL OR OF
        BAD ADDRESSES:  0,0
        LOGICAL AND OF
        BAD DATA:       777777,777777
        LOGICAL OR OF
        BAD DATA:       0,0
SYSTEM MEMORY CONFIGURATION:

CONTROLLER: #0  MA20  64 K
F0:     6000,0  F1:     36100,16012
        INTERLEAVE MODE:        4-WAY
        REQ ENABLED:    0 2
        LOWER ADDRESS BOUNDARY: 0
        UPPER ADDRESS BOUNDARY: 377777
        ERRORS DETECTED:        NONE
CONTROLLER: #1  MA20  64 K
F0:     6000,0  F1:     36100,16005
        INTERLEAVE MODE:        4-WAY
        REQ ENABLED:    1 3
        LOWER ADDRESS BOUNDARY: 0
        UPPER ADDRESS BOUNDARY: 377777
        ERRORS DETECTED:        NONE
CONTROLLER: #2  MA20  64 K
F0:     16000,0 F1:     36100,436012
        INTERLEAVE MODE:        4-WAY
        REQ ENABLED:    0 2
        LOWER ADDRESS BOUNDARY: 400000
        UPPER ADDRESS BOUNDARY: 777777
        ERRORS DETECTED:        ADDR PARITY
CONTROLLER: #3  MA20  64 K
F0:     16000,0 F1:     36100,436005
        INTERLEAVE MODE:        4-WAY
        REQ ENABLED:    1 3
        LOWER ADDRESS BOUNDARY: 400000
        UPPER ADDRESS BOUNDARY: 777777
        ERRORS DETECTED:        ADDR PARITY
ERRORS DETECTED DURING SWEEP:
ADDRESS DATA
```

4.10.1 Report Description

This entry is recorded in the error file each time the monitor receives an APR interrupt because of a parity error after all of physical memory has been scanned looking for more errors. If the original error also generated a page fail trap, the monitor will also create an entry for the processor parity trap. Complete details of the system recovery procedures may be found in Chapter 2 of this manual.

The CONI APR and ERA values reported are the contents of these registers at the time the first error occurred. The PC AT INTERRUPT value includes the flags in the left half. The BASE PHYsical Memory ADDRess AT FAILURE is from the right half of the contents of the ERA.

The NUMBER OF ERRORS on this sweep refers to the number of parity errors detected during this sweep of physical memory. If the value is zero, there were no errors detected and the LOGICAL AND function for both bad addresses and bad data will be 777777, 777777 and the LOGICAL OR functions will be 0, 0.

The section of this report labeled SYSTEM MEMORY CONFIGURATION lists the physical memory configuration and any detected errors at the time of the first error. These are the results of S-BUS DIAGNOSTIC FUNCTIONS for all memory controllers on this CPU.

The octal values for both FUNCTION 0 (F0:) and FUNCTION 1 (F1:) are listed as well as the text translations for these values.

The last section of this report lists the first 10 errors detected during the sweep if any errors were detected. Only the first 10 addresses and data values are listed as the logical and logical or data reported in the top section will identify any common bits in either address or data if more than 10 errors occur on the sweep.

4.11 SUMMARY REPORT

Each listing generated by SYSERR includes a SYSTEM SUMMARY at the end of the listing regardless of the listing control switches specified in the command string. This report will point out counts of all entries seen in the system error file within the date and time period specified in the command string. If the user, for example, was only interested in RP04 errors, this portion of the listing will also point out any other errors (such as memory parity) which may be of interest.

The following pages are a sample of one SYSTEM SUMMARY REPORT.

```
SYSTEM ERROR REPORT COMPILED ON THURSDAY, JANUARY 15, 1976 11:48:59      PAGE 263
              - SYSTEM SUMMARY FOR SYSTEM # 1031.
*******************************************************************************


FILE ENVIRONMENT
        SYSERR VERSION 5(230)
        INPUT FILES:    SYS:ERROR.SYS  CREATED: THU 15 JAN 76 11:42:40AM
        OUTPUT FILE:    DSK:TU45.LST
        SWITCHES:       /DEV:TU45/DETAIL: /BEGIN: 13-JAN-76 AT 11:46:11 /END: 14-JAN-76 AT 11:46:11
        DATE OF FIRST ENTRY PROCESSED:  MON  5 JAN 76 12:47:07AM
        DATE OF LAST ENTRY PROCESSED:   THU 15 JAN 76 11:42:40AM
        # OF INCONSISTENCIES DETECTED IN ERROR FILE:     18.

ENTRY OCCURANCE COUNTS
        TOTAL TOPS20 SYSTEM RELOADED(CODE 101):          5.
        TOTAL TOPS20 BUGHLT-BUGCHK(CODE 102):            20.
        TOTAL MASSBUS DEVICE ERROR(CODE 111):            433.
        TOTAL FRONT END DEVICE ERROR(CODE 130):          9.

TOPS20 BUGHLT-BUGCHK(CODE 102)
        BUGHLT/BUGCHK BREAKDOWN:
                ILLUUO  1.
                PH2DNA  7.
                OVRDTA  5.
                ILUST1  1.
                MTANOI  2.
                DIRB2S  1.
                DIRSY5  2.
                PWRRES  1.
```

```
SYSTEM ERROR REPORT COMPILED ON THURSDAY, JANUARY 15, 1976 11:49:ØØ     PAGE 264        - MASSBUS SYSTEM ANALYSIS(RH2Ø)

                        HARDWARE DETECTED
             PAR       LWC  SWC  RES       CHN  OVR
             ERR  EXC  ERR  ERR  ERR  ERR  RAE  RUN
MTØ5Ø HARD          1    1
      SOFT

MTØ51 HARD         22              2
      SOFT

MTØ52 HARD         15
      SOFT          8

MTØ64 HARD
      SOFT         59

MTØ65 HARD          1
      SOFT         16
```

```
SYSTEM ERROR REPORT COMPILED ON THURSDAY, JANUARY 15, 1976 11:49:01      PAGE 265        - MASSBUS SYSTEM ANALYSIS(RH20)


                             HARDWARE DETECTED
                PAR          LWC   SWC   RES          CHN   OVR
                ERR    EXC   ERR   ERR   ERR    ERR   RAE   RUN
DP110 HARD
      SOFT              1
```

```
SYSTEM ERROR REPORT COMPILED ON THURSDAY, JANUARY 15, 1976 11:49:Ø2      PAGE 266        *TU16 & TU45 DEVICE ANALYSIS
        ERROR TOTALS DETECTED BY DEVICE


       CC  U  O  D  N  I  F  N PL IV  D  F  P  R  I  I
       OR  N  P  T  E CF  C  S ER NP  P  M  A  M  L  L
       RC  S  I  E  F SM  E  G FC CE  A  T  R  R  R  F
                                      R

MTØ5Ø H
      S  1                       1

MTØ51 H
      S       13              7 22  9

MTØ52 H
      S 22                      22 23

MTØ64 H
      S 51             26       48  1  8

MTØ65 H
      S  2                       3  1 14
```

```
SYSTEM ERROR REPORT COMPILED ON THURSDAY, JANUARY 15, 1976 11:49:Ø3    PAGE 267        -RPØ4 DEVICE ANALYSIS

        ERROR TOTALS DETECTED BY DEVICE


                 - ERROR REG, #1 -                              - ERROR REG, #2 -                     - ERROR REG, #3 -
        D  U  O  D  W  I  A  H  H  E  W  F  P  R  I  I *A  P  3  I  N  M  W  F  T  T  M  C  W  C  W *O  S  D  A  P  U  V  P
        C  N  P  T  L  A  O  C  C  C  C  E  A  M  L  L *C  L  Ø  X  H  H  R  E  U  D  S  S  S  S  C *C  K  C  C  R  W  U  S
        K  S  I  E  E  E  E  R  E  H  F  R  R  R  R  F *U  U  V  E  S  S  U  N  F  F  E  U  U  F  U *Y  I  L  L  E  R  F  U
                             C                         *         U                                  *L
DP11Ø H
      S  1
```

```
SYSTEM ERROR REPORT COMPILED ON THURSDAY, JANUARY 15, 1976 11:49:04      PAGE 268

         FRONT END DEVICE ERROR SUMMARY:
         CD20:   2.
         KLCPU:  1.
```

4.11.1 Report Description

The first page of the report is always listed and shows the input file, output file, command switches, and the counts of all entry types found in the error file during the time period specified in the command string. The first page may also include a breakdown of the BUGHLT-BUGCHK entries, if any were detected.

The following pages list summaries for the Massbus devices and their controllers. Only those devices are listed for which errors were detected. The first pages list the counts for each error detected by the controller. The maximum value listed for any type error is 9999. The following pages list the same type summary for the error registers for each type Massbus device. Here the maximum number listed is 99.

The last page of the summary lists a breakdown of the front end device errors detected in the file. The sum of these numbers may be different than the value listed on the first page of the summary for the FRONT END DEVICE ERROR (CODE 130): entries. This is because 1 KLERROR will require several error file entries to contain all the information.

CHAPTER 5

TROUBLESHOOTING DECSYSTEM20 WITH SYSERR

(How To Use SYSERR)

To Be Supplied

APPENDIX A

ERROR MESSAGES

Error messages from SYSERR are of two types, FATAL and WARNING. Fatal errors are preceded by a question mark (?), warning messages are preceded by a percent sign (%).

The following fatal errors are currently output from SYSERR:

?ENTER ERROR ON OUTPUT FILE

> This implies that SYSERR is unable to do an enter on device and directory specified by user. The user should check for disk quota exceeded or file protection failures.

?ERROR DURING OUTPUT

> SYSERR detected an output error while writing the list file.

?CAN'T OPEN INPUT DEVICE

> This implies that SYSERR was unable to perform an open on the input device specified by the user.

?CAN'T OPEN OUTPUT DEVICE

> This implies that SYSERR was unable to perform an open on the output device specified.

?LOOKUP ERROR ON INPUT FILE

> This implies that SYSERR was unable to lookup the input file specified by the user. Check to see that the input file is on the input device and is not read-protected.

?SYSERR TRYING TO DO LISTING

> This error message indicates that the first high segment, SYSERR, is about to try to produce a listing, however, none of the known entry codes are processed by this segment.

The following warning messages are output by SYSERR:

%EOF MARKER FOUND IN BODY OF SYSTEM ERROR FILE

SYSERR has seen an EOF word written by the monitor in the body of the error file. This is normal if error files are combined.

%DUMPING UNKNOWN ERROR TYPE IN OCTAL

SYSERR detected an entry whose error code did not match any of the known error types.

%EXCEEDED PAGE LIMIT . . . . PERFORMING SUMMARY

SYSERR has output more than the allowable pages of listing, currently defaulted to 1000, and is now terminating listing and performing summary.

The reason for this limit is that 1000 pages of report is more than anyone can absorb. A repeated error can generate a lot of repeated output. The user should examine the summary and select the subset by date and device that he is interested in.

%ENTRY WITH ZERO LENGTH HEADER SPECIFIED

%ENTRY WITH ZERO LENGTH BODY SPECIFIED

Both of these messages are indications usually that SYSERR has lost sync in the error file. The recovery is attempted as started in Appendix B.

%SYRERI:FATAL ERROR READING INPUT FILE

SYSERR has encountered a checksum or parity error while reading the current input file. The package will look to see if any other input files process them, and then generate the summary listings.

%UNKNOWN DEVICE NAME FOUND IN ENTRY

SYSERR has found a device name in the error file it doesn't recognize such as DPA7 if SYSERR's configuration only knows about 6 DPA's. SYRUNV should be changed to reflect your system configuration. See Appendix C for instructions concerning compiling and loading.

%EXPECTED ERROR CODE NOT FOUND ON TABLE OF SUBJECT ERROR CODES

An event code has been found in the error file within the range of those codes eligible for SYSERR processing (See Appendix B) but none of the SYSERR modules have the ability to process it. The entry will be dumped in octal in the output file.

%SYRRNR: RESTARTING IN THE NEXT BLOCK OF ERROR FILE

SYSERR has encountered problems and has lost sync in the current block. It has gotten the next block of the file, found the offset and has started processing again with the first entry in this next block.

%SYRCNR: CANNOT RE-SYNC, TRYING NEXT BLOCK

As above, SYSERR has lost sync and gotten the next block of the file but there is no pointer word to the start of the first entry in this block. SYSERR will look at each block until either it finds a valid pointer word or end-of-file is encountered.

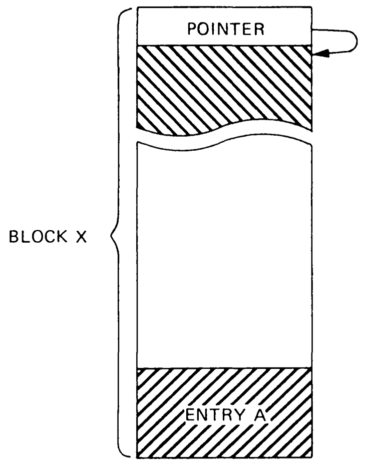
%DUMPING PARTIAL CONTENTS OF KLERROR FILE IN OCTAL

SYSERR was building the file in core and either of two events occurred: 1) an inconsistency was detected in the system error file, or 2) SYSERR detected the start of another KLERROR file.

APPENDIX B

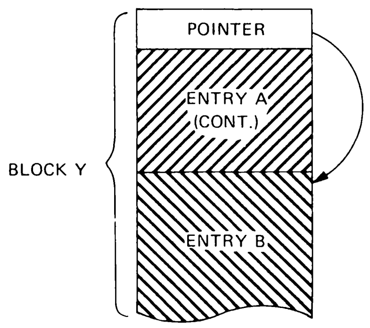
ERROR FILE DESCRIPTIONS

This appendix contains descriptions of the format of the error file and contents of each type of entry. The file is created and appended to by a portion of the monitor and read by SYSERR. Each entry is considered a separate entity by SYSERR and is treated separately. The recording program also considers each entry or record separately and appends each to the end of the file. The only exception to this policy is the synchronization word at the start of each block (128 words). This word is a pointer or offset to the start of the first entry in the current block and is used by SYSERR to get back in sync in case of trouble. This word is required because entries may cross block boundaries to conserve disk space. The use of this resync word is described in Appendix A, SYSERR ERROR MESSAGES and the following diagram shows the typical layout of entries across a block boundary.



Pointer in BLOCK X points to next word as the start of first entry in this block.

The last entry in this block (ENTRY A) crosses into BLOCK Y.



The pointer in BLOCK Y points to the start of ENTRY B.

Each entry or record in the error file is composed of two sections, a header section and a body section. The header section contains the entry type, date and time the event was recorded, the processor serial number which detected the error, and the length of the header and body sections. The body section contains the various data items which make up the entry. The format of the header section is constant for each version of the header section regardless of the entry type and is described below. The format of the body section for each entry type is described on succeeding pages.

ENTRY HEADER FORMAT

This header is used to describe the contents of each entry.

HDRCOD	*
HDRDAT	Date & time of entry in Universal Format
HDRUPT	System uptime at entry. LH = #days, RH = fraction of day
HDRPSN	Processor serial # where entry was recorded

	BITS	DESCRIPTION
*HDRCOD	0-8	Entry type, tells program how to process this entry. See below for range of event codes.
	9-16	Reserved.
	17	This entry recorded by TOPS20.
	18-23	Header Format Version, presently = 1.
	24-26	Header Length, presently = 4.
	27-35	Entry Length excluding header, maximum = 777.

Event Codes

All event codes are in the range of 0 to 777 with reservations as described:

000	Illegal
1-376	Reserved by DEC for use with SYSERR.
400-477	Reserved for customer use with SYSERR.
500-577	Reserved by DEC for use with programs other than SYSERR.
600-677	Reserved for customer use with programs other than SYSERR.
700-777	Reserved for all for error file control.

In some cases (mostly for error file control) only the first word (HDRCOD) is included. This is the minimum required for any entry. The current event codes used for file control are as follows:

377	The recording program has detected an error in the file and has started using the next sequential error file. See Appendix A of this manual.
775	Offset word in the first word of each block of the error file. RH points to start of the first entry in this block.
777	End of File, tells SYSERR to look for next file or to start summary listings if no other files are found.

SYSTEM RELOADED EVENT CODE 101

```
SEC%RL==101             ;Event code
  RL%SVN==0             ;System name (ASCIZ PTR)
  RL%STD==1             ;Time of system build (universal FMT)
  RL%VER==2             ;System version number
  RL%SER==3             ;APR serial number
  RL%OPR==4             ;Operator answer to why reload (ASC PTR)
  RL%HLT==5             ;Bughlt address (if auto reload)
  RL%FLG==6             ;Flags

RL%SIZ==7               ;Size of data block
RL%LEN==RL%SIZ+30       ;Size of whole block (incl 2 strings)
```

```
;BUGHLT/BUGCHK EVENT CODE 102

SEC%BG==102               ;Event code
  BG%SVN==0               ;System name (ASCIZ)
  BG%SER==1               ;APR serial number
  BG%VER==2               ;Monitor version
  BG%SDT==3               ;TAD of Monitor build
  BG%FLG==4               ;Flags
    BG%CHK==1B1           ;BUGCHK type code
    BG%INF==1B2           ;BUGINF type code
    BG%HLT==1B3           ;BUGHLT type code
  BG%ADR==5               ;Address of HLT/CHK
  BG%JOB==6               ;FORKX,,job number
  BG%USR==7               ;User number
  BG%PNM==10              ;Program name (sixbit)
  BG%MSG==11              ;Message (ASCIZ)
  BG%ACS==12              ;ACS
  BG%PIS==32              ;PI status
  BG%RCT==33              ;Register count
  BG%REG==34              ;Registers (maximum of 4)
  BG%NAM==40              ;Sixbit name of check
  BG%DAT==41              ;Time and date of BUGHLT/BUGCHK
  BG%CNT==42              ;Number of bug checks since startup
BG%SIZ==43                ;Size of data Block
BG%LEN==BG%SIZ+30         ;Length of total block, incl 2 strings
```

MASSBUS DEVICE ERROR EVENT CODE 111

```
SEC%MB==111             ;Event Code
  MB%NAM==0             ;Device name (if available)
  MB%VID==1             ;Volume ID (sixbit)
  MB%TYP==2             ;Channel..device type - see PHYPAR
  MB%LOC==3             ;Location of error - sector or file..record
  MB%FES==4             ;Final error state - device dependent
  MB%CNI==5             ;CONI initial
  MB%CIF==6             ;CONI final
  MB%SEK==7             ;Number of seeks
  MB%RED==10            ;Number of blocks/frames read
  MB%WRT==11            ;Number of blocks/frames written
  MB%UAD==42            ;Unit address
  MB%SPE==43            ;Soft Positioning errors
  MB%HPE==44            ;Hard positioning errors
  MB%OVR==45            ;Overruns
  MB%ICR==46            ;Initial TCR
;The following locations are the units Massbus registers in order
;Final contents,,initial error contents
  MB%REG==47
MB%SIZ==MB%REG+20       ;Size of data block
MB%LEN==MB%SIZ          ;Total length, currently no strings    reported
```

FRONT END ERRORS EVENT CODE 130

```
SEC%FE==130             ;Event code
  FE%FJB==0             ;Fork number,,job number
  FE%DIR==1             ;Directory numbers
  FE%ID==2              ;Front end software version
  FE%PGM==3             ;Sixbit name of program
  FE%COD==4             ;Protocol device code (1B0=unknown)
  FE%PRT==5             ;-Length of data,,start of data
  FE%DTE==6             ;DTE number
  FE%INF==7             ;Start of error information
FE%SIZ==7               ;Size of data block (header)
FE%LEN==FE%SIZ          ;Minimum block to allocate
```

FRONT END RELOAD ENTRY. GIVES -11 REBOOT INFORMATION

EVENT CODE 131

```
SEC%11==131              ;-11 Reload
  R1%NUM==0              ;-11 number
  R1%STS==1              ;Reload status bits
        .R1GTF==1B0      ;GTJFN failed for dump file
        .R1OPF==1B1      ;OPENF failed for dump file
        .R1DPF==1B2      ;Dump failed
        .R110E==1B3      ;To -10 error on dump
        .R111E==1B4      ;To -11 error on boot
        .R1ASF==1B5      ;ASGPAG failed on dump
        .R1RLF==1B6      ;Reload failed
        .R1DPF==1B7      ;-11 didn't power down
        .R1PUF==1B8      ;-11 didn't power up
        .R1RMF==1B9      ;ROM did not ack the -10
        .R1BSF==1B10     ;-11 boot program didn't make it to the -11
        .R1NRL==1B11     ;11 took more than 1 minute to reload.
                         ;will cause a retry
        .R1RTC==6B35     ;Retry count
 R1%FNM==2               ;File name pointer
R1%SIZ==3                ;Number of entries
R1%LEN==R1%SIZ+^D20      ;Allow long string
```

PROCESSOR PARITY TRAP EVENT CODE 160

```
SEC%PT==160              ;Event code
  PT%PFW==0              ;Page fail word
  PT%BDW==1              ;Bad data word
  PT%GDW==2              ;Good data word
  PT%USR==3              ;User number
  PT%JOB==4              ;FORKX,,JOBN
  PT%PGM==5              ;Program name (sixbit)
  PT%PMA==6              ;Physical memory address
  PT%TRY==7              ;Flags,,retry count
        PT%HRD==1B1      ;Hard error
        PT%CCF==1B2      ;Cache failure
        PT%CCH==1B3      ;Cache in use
        PT%ESW==1B4      ;Errors on sweep to core
PT%SIZ==10               ;Size of data block
PT%LEN==TP%SIZ           ;Length of total block
```

PROCESSOR PARITY INTERRUPT EVENT CODE 161

```
SEC%PI==161             ;Event code

  PI%CNI==0             ;CONI APR

  PI%ERA==1             ;ERA

  PI%FP2==2             ;PC

  PI%SWP==3             ;Number of errors this sweep

  PI%AAD==4             ;Logicl and of bad addresses

  PI%OAD==5             ;Logical or of bad addresses

  PI%ADA==6             ;Logical and of bad data

  PI%ODA==7             ;Logical or of bad data

  PI%SBD==10            ;Logical SBus diag function data (10 wds)

          PI%NSD==^D10 ;Number of SBus diag fn words

  PI%ADD==22            ;First 10. bad addresses

  PI%DAT==34            ;First 10. bad data words

          PI%NBW==^D10 ;Number of bad words

  PI%SIZ==46            ;Size of data block

  PI%LEN==PI%SIZ        ;Length of total block
```

APPENDIX C

ASSEMBLY INSTRUCTIONS FOR SYSERR PACKAGE

The SYSERR package for DECsystem-20 is comprised of 4 source modules:

SYRUNV.MAC	Universal file containing revision history, macro definitions, and low segment data area definitions and storage locations, etc. This module is only used during the assembly process and is not used at run time.
SYSERR.MAC	Routines for file initialization and command parsing.
SYSERD.MAC	PROCSD routines used for listing DECsystem-20 entries.
SYSERS.MAC	Summary listing routines for all entries.

Additional routines required to load with the package include:

SCAN.REL	Command scanner, general utility and output routines.
HELPER.REL	Finds and lists the HELP file.

A separate file which gives brief instructions for running SYSERR is SYSERR.HLP and should be located in the same directory as the SYSERR package.

The easiest method to compile, load, and save the SYSERR package is to use the batch control file distributed with the package. To submit the job to batch, the command is:

```
     @ SUBMIT SYSERR/RESTART:1/TIME:20:00/UNIQ:0
```

If CREF listings are desired add /TAG:CREF to the command. The control file may also be listed and the commands typed on your terminal as they appear in the file if you don't wish to use batch. The package should always be loaded with local symbols to allow debugging if required without having to re-compile or re-load the package.

INDEX

Error Detection, Recovery and
Reporting Reference Manual
EK-SEDRR-RF-001

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form.

Did you find errors in this manual? If so, specify by page.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name ______________________ Date ______________

Organization ______________________

Street ______________________

City ______________ State ______________ Zip Code or Country ______________

If you require a written reply, please check here. ☐

Please cut along this line.

-- Fold Here --

-- Do Not Tear - Fold Here and Staple --

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Communications
P. O. Box F
Maynard, Massachusetts 01754

digital